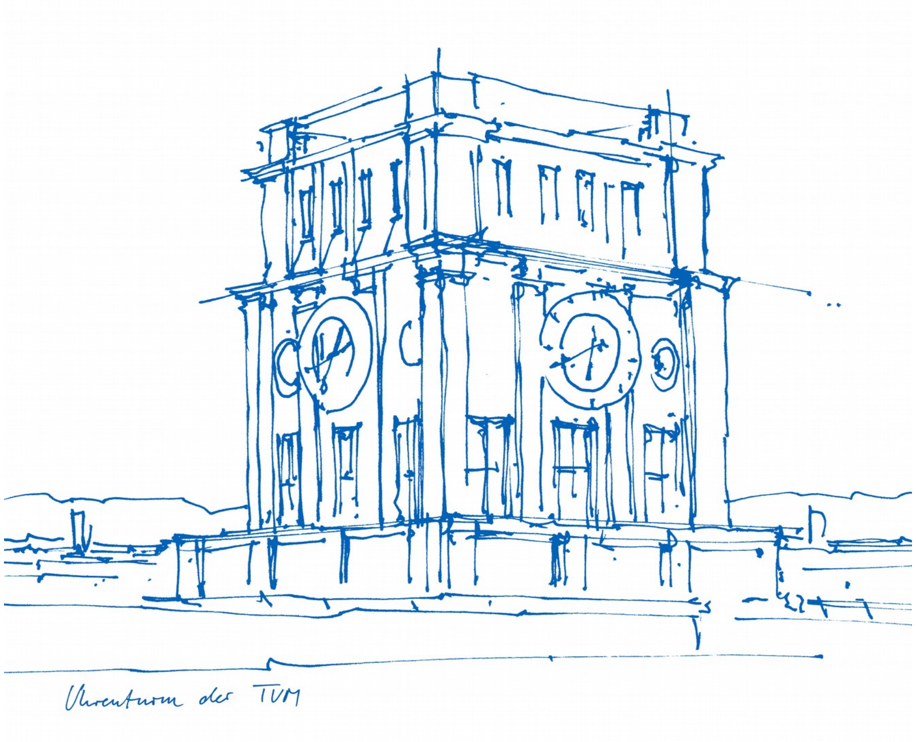
Chair III: Database Systems, Professorship for Data Mining and Analytics
Chair XXV: Data Science and Engineering
Department of Informatics
Technical University of Munich

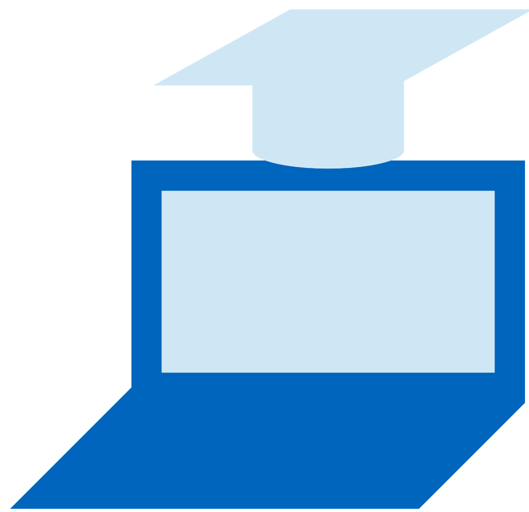
In-Database Machine Learning: Using Gradient Descent and Tensor Algebra

Maximilian E. Schüle, Frédéric Simonis, Thomas Heyenbrock,
Alfons Kemper, Stephan Günnemann, Thomas Neumann
Rostock, 04. März 2019

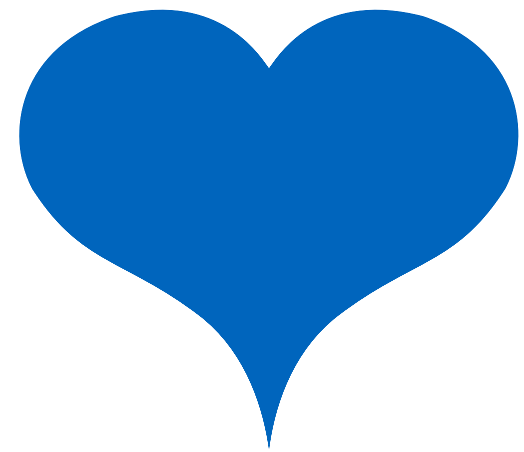


What Need Database Systems for ML?

Database Systems



Machine Learning



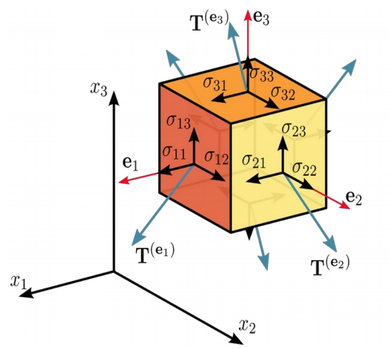
Why don‘t use HyPer?

What Need Database Systems for ML?

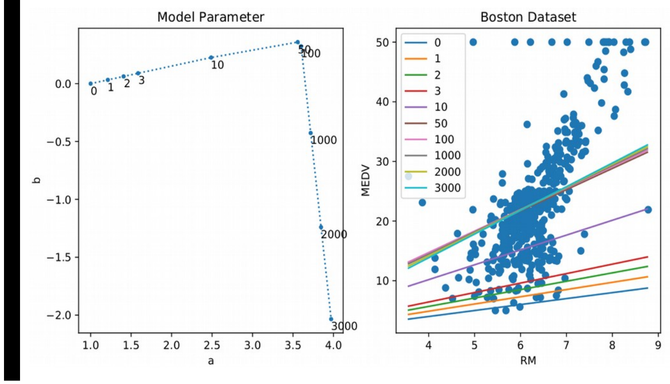
Machine Learning: data in tensors and a parametrised loss function

HyPer

Tensors



Gradient Descent



Advantages: Optimisation problems are solvable in the core of database servers
Goal: Make database systems more attractive

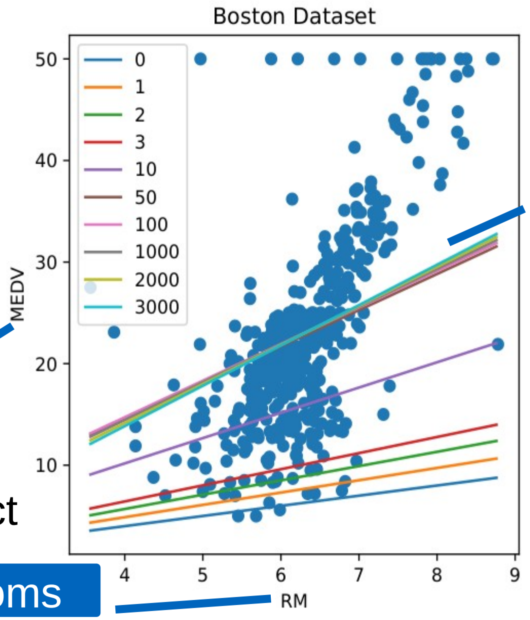
What it is: Architectural blueprint for the integration of optimisation models in DBMS
What it is not: Study about the quality of different optimisation problems

What is Gradient Descent?

Median Value

predict

#Rooms



Linear Regression

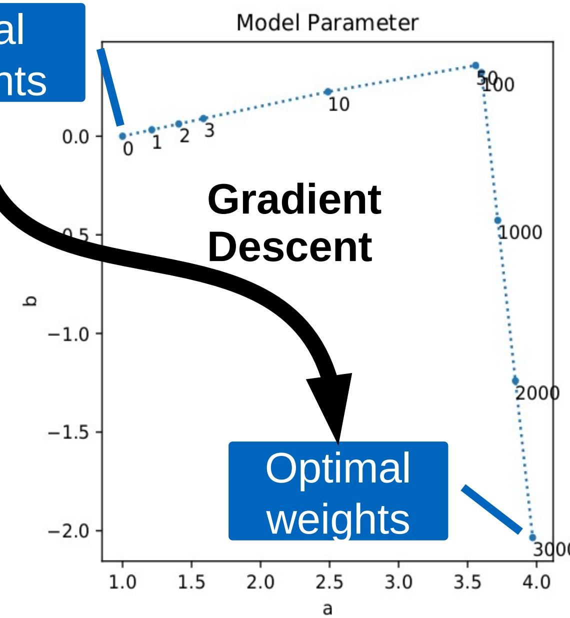
$$m_{a,b}(rm)=a*rm+b\approx medv$$

Optimal weights?

Gradient Descent!

$$l_{rm,medv}(a,b)=(m_{a,b}(rm)-medv)^2$$

Initial weights



Gradient Descent

Optimal weights

Training Data		**Test Data**	
RM	**MEDV**	**RM**	**MEDV**

How to optimse weights?

How to label data?

Approach

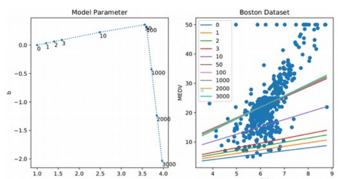
HyPer

Integration as **operators** in relational algebra

Representation of **mathematical functions** on relations

Concept of **pipelines**

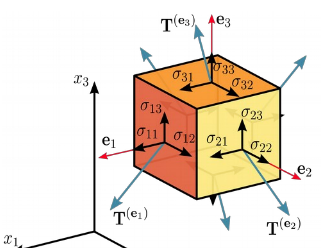
Gradient Descent



Gradient needed

Automatic differentiation

Tensors



Representation of tensors

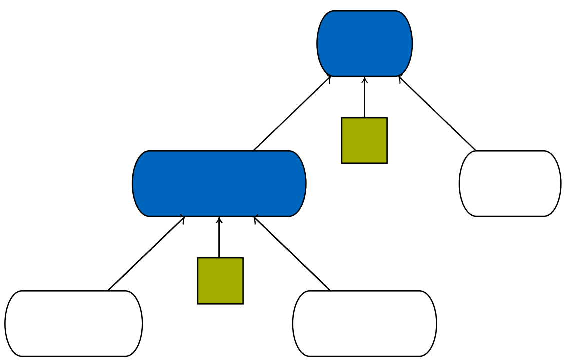
Either: one relation represents one tensor

Or: own tensor data type

Integration in Relational Algebra

Operator Tree

Operators for labelling and gradient descent: Pipelines (Weights/Data)



Model / Loss Function

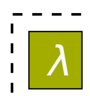
Representation of a loss- as well as of a model function

model function *m*

$$m_w(x) = \sum_{i \in m} x_i * w_i \approx y$$

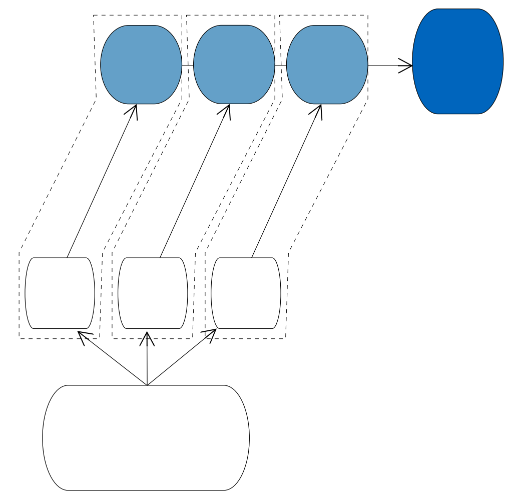
loss function *l*

$$l_{x,y}(w) = (m_w(x) - y)^2$$



Pipelining

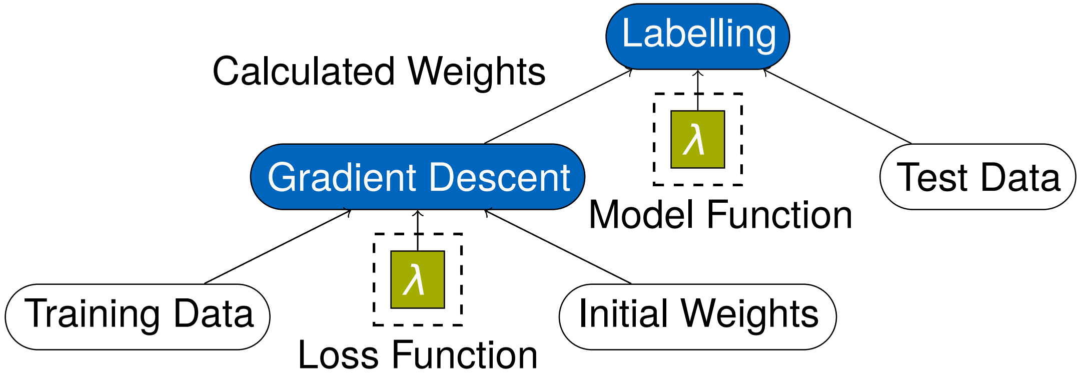
Integration as a pipeline breaker



Integration in Relational Algebra: Operator Tree

Two Operators needed

Gradient descent to optimise weights of a parametrised loss function
Labelling operator to label predicted values



Gradient Descent

Initial weights and training data as input and optimised weights as output
Lambda expression as loss function to be optimised

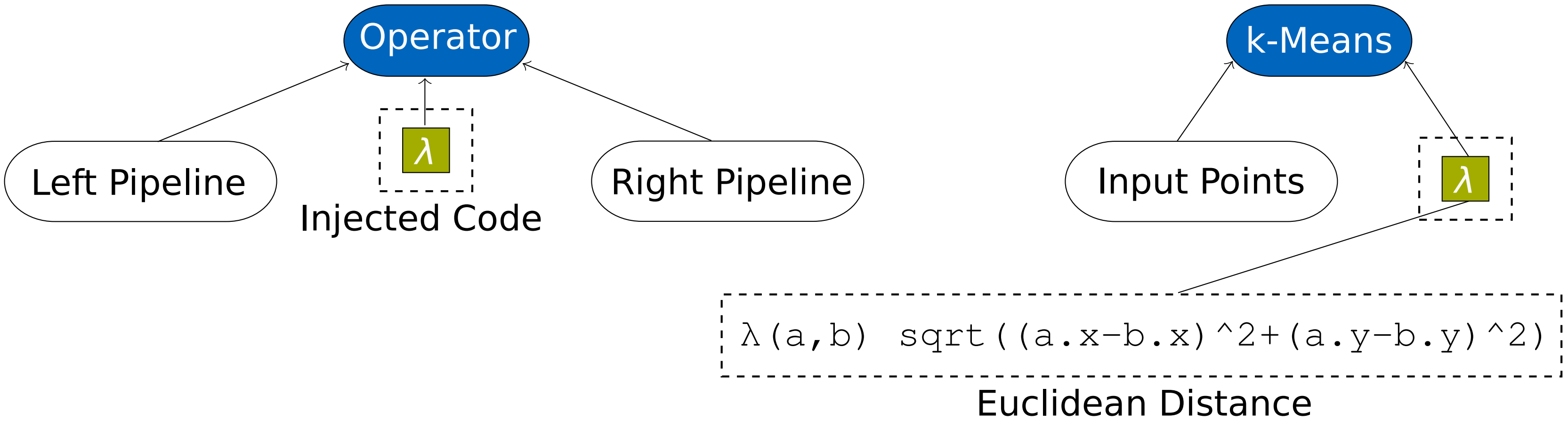
Labelling

Input: test dataset and optimal weights
Label: evaluated lambda expression for each tuple

Integration in Rel. Algebra: Lambda Functions

Lambda Expression

To inject user-defined code




```
select * from kmeans((table points),λ(a,b) sqrt((a.x-b.x)^2+(a.y-b.y)^2),2)
```

Integration in Rel. Algebra: Lambda Functions

	Notation	Relations/Lambda Functions
Weights	$w=(w_{1,} w_{2,\ldots}, w_m)$	`W{[w_1,w_2,…,w_m]}`
n tuple with *m* attributes	$x=(x_{1,} x_{2,\ldots}, x_m, y)$	`X{[x_1,x_2,..,x_m,y]}`
Model function	$m_w(x)=\sum_{i\in m} x_i * w_i \approx y$	`λ(W,X)(W.w_1*X.x_1+...+X.x_m)`
Loss function	$l_{x,y}(w)=(m_w(x)-y)^2$	`λ(W,X)(W.w_1*X.x_1+...+X.x_m-y)²`

Lambda Functions in SQL

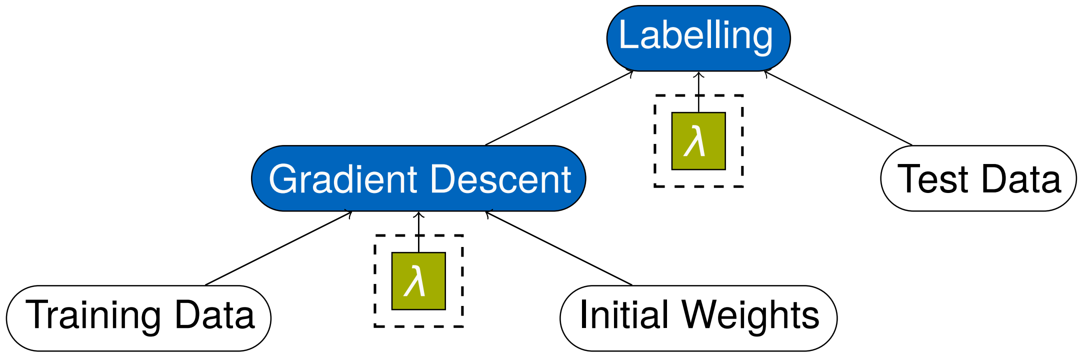
```
create table trainingdata (x float, y float);
create table weights(a float, b float);
insert into trainingdata… insert into weights…

select * from gradientdescent(
 - loss function as λ-expression
 λ(data, weights)(weights.a*d.x+weights.b-d.y)²,
 -- training set and initial weights
(select x,y from trainingdata d),
(select a,b from weights),
 -- learning rate and max. number of iteration
 0.05, 100
);
```

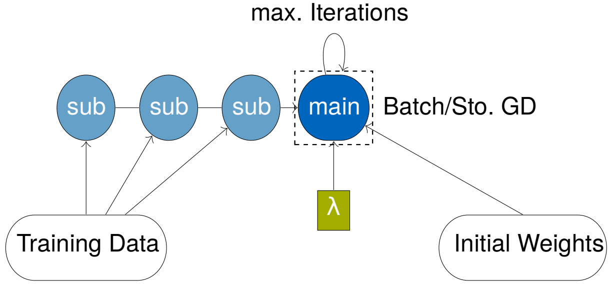
```
create table testdata (x float);
create table weights(a float, b float);
insert into trainingdata…
insert into weights…

select * from labeling(
 -- model function as λ-expression
 λ(data, weights)(weights.a*d.x+weights.b),
 -- training set and initial weights
(select x,y from testdata d),
(select a,b from weights)
);
```

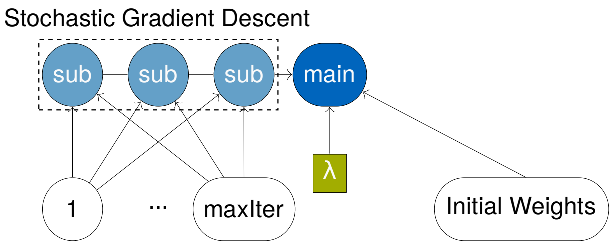
Integration in Relational Algebra: Pipelining



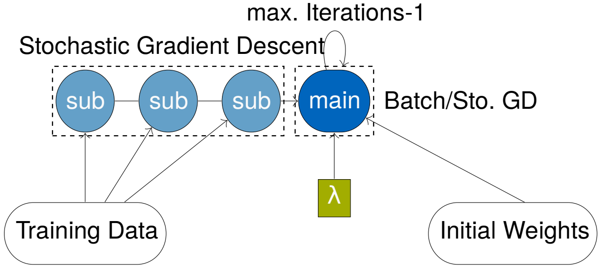
Materialising



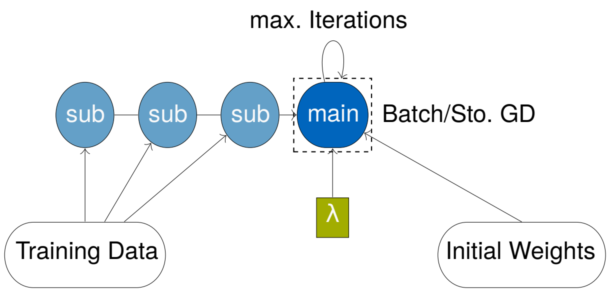
Pipelined



Combined



Integration in Relational Algebra: Pipelining

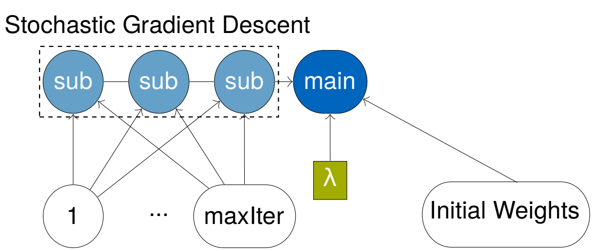


Materialising

Materialisation of all tuples (parallel/serial)

Any optimisation method possible

Parallelism: *parallel_for*



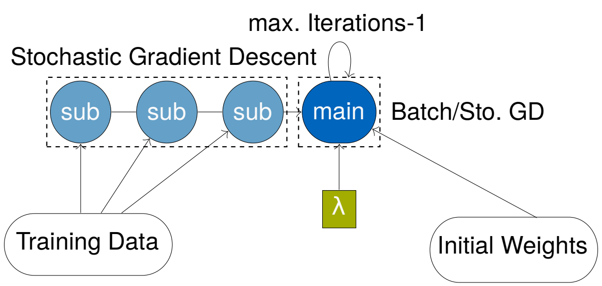
Pipelined

No materialisation

Stochastic gradient descent only

Distribution to pipelines

Downside: multiple copys of the operator tree



Combined

First iteration in pipelines
Remaining ones in the main thread

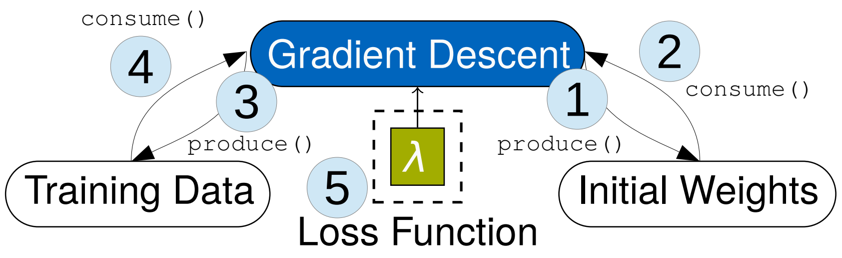
Automatic Differentiation for Gradient Descent

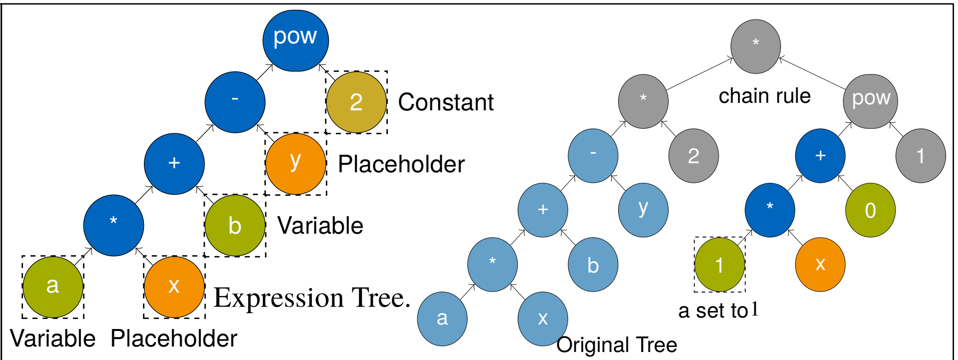
Need of a gradient for gradient descent: Automatic differentiation necessary

HyPer compiles SQL before execution

→ precompilation of the gradient, evaluation for each tuple using placeholders

Compilation





Execution

```
auto status = model_optimizer->trainable.train(
   ValuedNodes{model_gradient->model.placeholders, tensors});
```

Tensor Data Type

Extension of the PostgreSQL array data type

transpose $(t^t)_{i_1 i_2 i_3 \dots i_m} = t_{i_2 i_1 i_3 \dots i_m}$

addition/subtraction/scalar product $(t+s)_{i_1 i_2 \dots i_m} = t_{i_1 i_2 \dots i_m} + s_{i_1 i_2 \dots i_m}$

multiplication (inner tensor product) $T \in \mathbb{R}^{I_1 \times \dots \times I_m = o}, U \in \mathbb{R}^{J_1 = o \times \dots \times J_n}, \quad s_{i_1 i_2 \dots i_{m-1} j_2 \dots j_m} = \sum_{k \in [o]} t_{i_1 i_2 \dots i_{m-1} k} u_{k j_2 \dots j_m}$

Linear Regression

$$w = (w_1, w_2, \dots, w_m)$$

$$x = (x_1, x_2, \dots, x_m)$$

$$y = (y_1, y_2, \dots, y_n)^t$$

$$w = (X'^T X')^{-1} X'^T y$$

Linear Regression in SQL with Tensors

```
select (array_inverse(array_transpose(x)*x))*(array_transpose(x)*y)
from (
   select array_agg(x) x
   from (
      select array[1,x_1,x_2] as x
      from datapoints) sx
   ) tx, (
      select array_agg(y) y
      from (
         select array[y] y
         from datapoints
      ) sy
   ) ty;
```

Evaluation

Evaluation

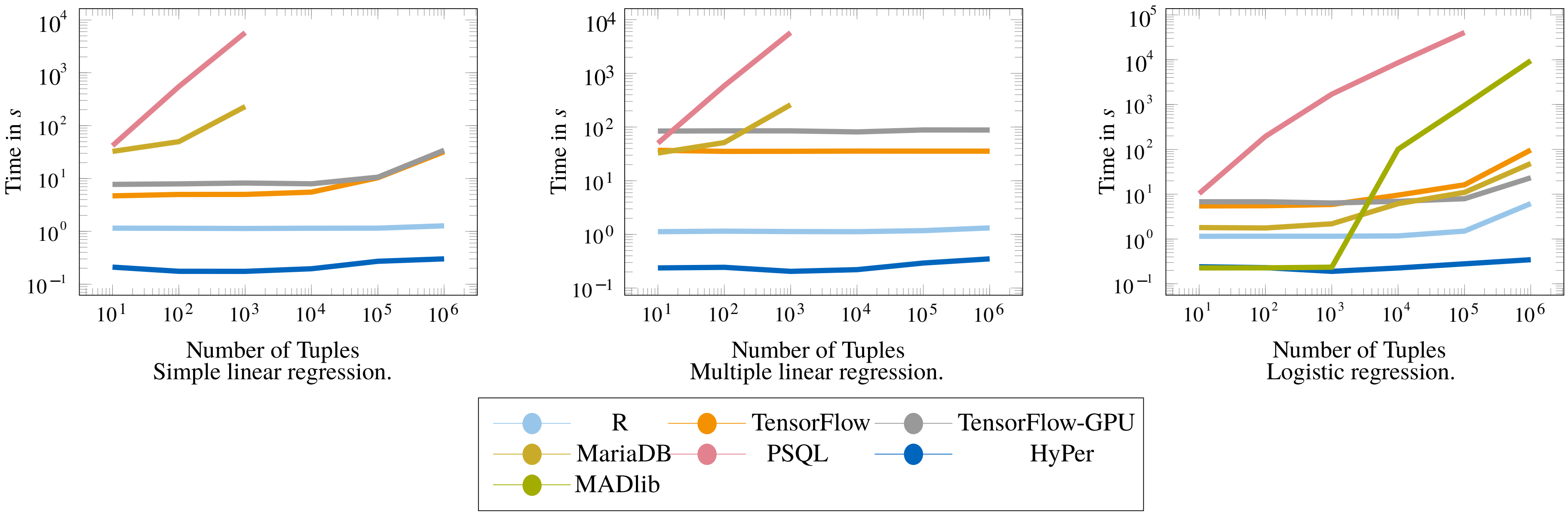
Tools
HyPer, MariaDB 10.1.30,
PostgreSQL 9.6.8 with MADlib v1.13,
TensorFlow 1.3.0, R 3.4.2

Machine
Intel Xeon E5-2660 v2 CPU (20x 2.20 GHz)
256 GB DDR4 RAM
Nvidia GeForce GTX 1050 Ti

Data
Chicago Taxi Rides Dataset (10^6 Tupel)

Tests
Linear regression (2-3 attributes)
Logistic regression (2 attributes)
k-Means clustering

Evaluation – Runtimes of GD



Simple linear regression.

Multiple linear regression.

Logistic regression.

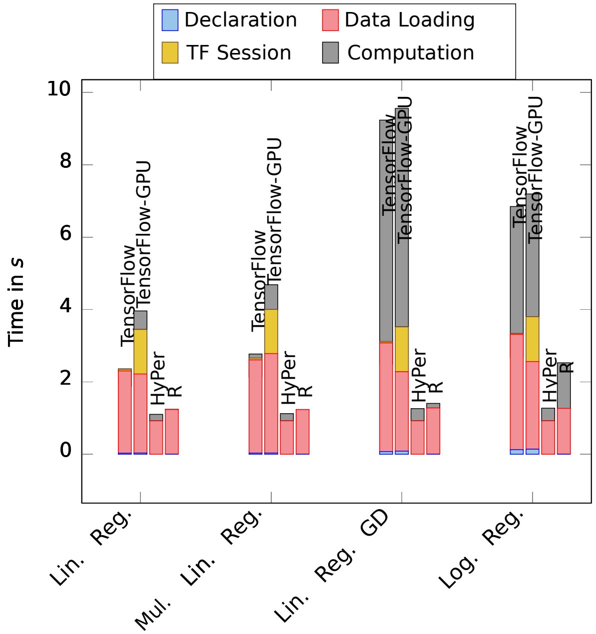
Runs

5000 iterations

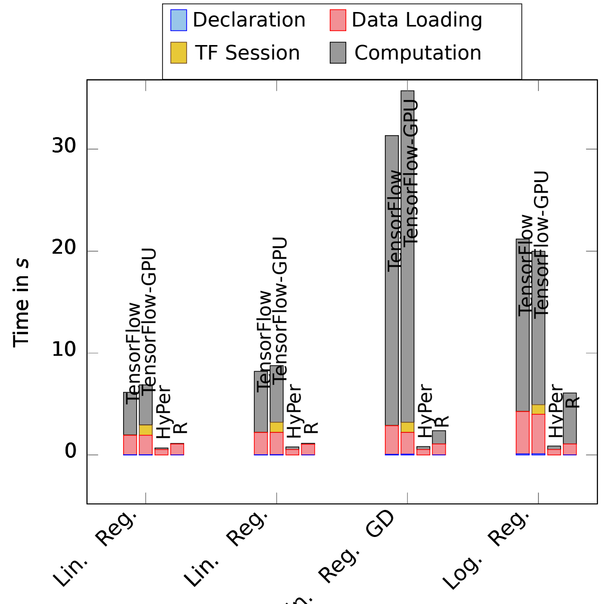
Database systems: no time for data loading needed

HyPer faster, PSQL and MariaDB (using procedures) slower

Evaluation – Ratio Computation/Loading Time



(a) 10^5 tuples.



(b) 10^6 tuples.

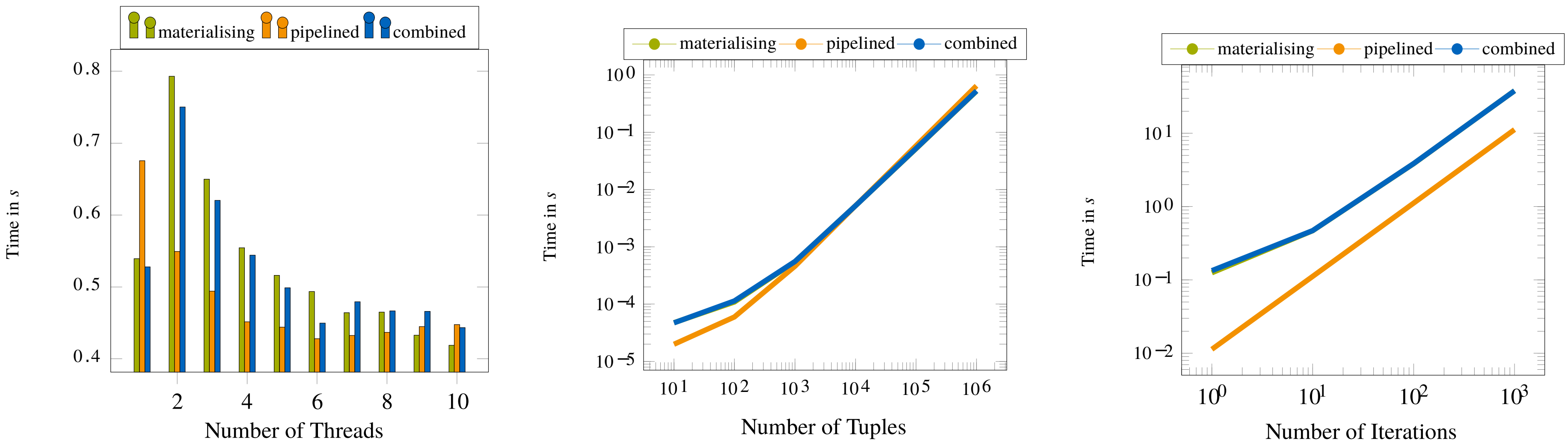
Runs

parameters: 10 iterations, 10^6/10^7 tuples

Most of the time: data loading

Not necessary, when computation is done inside of the database system

Evaluation – Architectures



(a) Scale Tests. (b) Varying the input size. (c) Varying the number of iterations.

Evaluation of the architectures: materialising, pipelined, combined

Standard parameters: 10 iterations, 10^6 tuples, one thread

Observation

Pipelined faster, but only allows stochastic GD and needs fixed number of iterations

All implementations scale

Combined plan: low

Conclusion

Database systems: more computations (tensors + gd)

Aim of the work

Saving time by moving ML operations into the core of DBMS

Gradient descent and labelling in SQL + Lambda

Different architectures for gradient descent

Future Work

Support of tensor datatypes

- Second view on relations: combining SQL and ArrayQL

Generic language for machine learning

- Dedicated language that compiles to SQL
- Embedding of Python or R in SQL